

METHOD AND SYSTEM FOR PROVIDING VERSION CONTROL OF PARAMETERS IN A COMMAND-BASED API USING JAVA SERIALIZATION

BACKGROUND OF THE INVENTION

Field of the Invention

[0001] The invention relates generally to the field of computer systems and, more specifically, to a technique for determining whether a server-side host supports a command provided by a client-side host.

Description of the Related Art

[0002] Computer systems include host computers that communicate with one another via a network to run network applications. Typically, software is distributed and run on two or more hosts to realize the application. The network applications have application-layer protocols that define the format and order of the messages that are exchanged between the hosts, and what actions to take when a message is transmitted or received. In particular, a network application typically includes a client side and a server side. In this case, the application may be referred to as a client/server application. A client side on one host may communicate with a server side on another host. The client is usually the host that initiates a communication or session with another host.

[0003] However, difficulties arise when the client host and server host are running different versions of the network application or other software. In this case, the client may provide a command to the server requesting it to perform a specific function that the server does not support. Various approaches have been developed to address this problem. For example, the client may send additional data to the server such as a codeword that identifies the version the client is using. However, this approach lacks

generality and requires that a special protocol for coding and decoding the version data be implemented.

BRIEF SUMMARY OF THE INVENTION

[0004] To overcome these and other deficiencies in the prior art, the present invention provides a technique for determining whether a server supports different functions that are represented by parameter objects in a command that is sent to a server-side host by a client-side host.

[0005] In a particular aspect of the invention, a method for providing a command from a client-side host to a server-side host is provided. The method includes invoking a client-side application programming interface (API) at the client-side host to pass in a set of parameter objects, and to provide a command object that contains the parameter objects; wherein each of the parameter objects represents a different parameter of a command; serializing the command and parameter objects to provide serialized command and parameter objects; and communicating the serialized command and parameter objects to the server-side host as the command.

[0006] In another aspect of the invention, a method for processing a command from a client-side host at a server-side host is provided. The method includes receiving serialized command and parameter objects at the server-side host as a command from the client-side host; wherein the command object contains the parameter objects, and each of the parameter objects represents a different parameter of the command; and deserializing the serialized command and parameter objects to determine whether the server-side host is compatible with the different parameters represented by the parameter objects.

[0007] A related program storage device is also provided.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] These and other features, benefits and advantages of the present invention will become apparent by reference to the following text and figures, with like reference numbers referring to like structures across the views, wherein:

[0009] Fig. 1 illustrates a client-side host and a server-side host in a computer system; and

[0010] Fig. 2 illustrates a method for providing version control.

DETAILED DESCRIPTION OF THE INVENTION

[0011] Generally, the present invention provides a technique for providing version control of parameters in a command-based, client-server network application programming interface (API). In a client-server network API, the levels of functionality supported by the client and server may differ. Moreover, in a command-based API, new functionality may be represented as new additional parameters to the commands. For example, for a server host in a storage subsystem, the functions may be storage-related. The functions may specify, e.g., a copy type to be performed, a location to store data, or other storage action to be taken. For instance, in the IBM® Enterprise Storage Server (ESS) storage subsystem, one function may be a normal, continuous flash copy (an instant copy of data, such as a point-in-time copy of a volume), while another function may be an incremental flash copy. The functions may also be thought of as optional features that may or may not be supported by different server hosts. The programming challenge is how to determine if a specific parameter of a command from the client portion of the API is supported by, e.g., compatible with, the server portion of the API. The present invention addresses this challenge by using Java deserialization of class type to determine if the server supports a specific parameter and the function represented by the parameter.

[0012] Fig. 1 illustrates a client-side host 100 and a server-side host 150 in a computer system. The client host 100 includes a processor 110, memory 105 and network interface 115, while the server host 150 similarly includes a processor 160, memory 155 and network interface 165. The network interfaces 115 and 165 communicate with one another via a network 130 such as the Internet. The client-side host 100 may implement a client-side API that comprises a client-side of a command-based API, while the server-side host 150 may implement a server-side API that comprises a server-side of the command-based API. In a command-based API, one or more hosts send commands to one or more other hosts. For example, a command-based API may be used by a storage subsystem host and remote hosts that send data to the storage subsystem host to be backed up. The remote hosts may issue commands to the storage subsystem host to instruct it to backup their data, or to retrieve previously backed up data, for instance.

[0013] The general operation and configuration of the processors 110, 160, memories 105, 155 and network interfaces 115, 165 is well known in the art and is therefore not described in detail. The components illustrated are provided to assist in understanding the invention. The hosts 100, 150 may be general-purpose computers, workstations, servers, portable devices such as PDAs, or other computer devices. The functionality described herein can be achieved by configuring the hosts 100 and 150 with appropriate software. In one approach, the software comprises an object-oriented software such as Java code that is stored in the memories 105, 155. In this case, the memories 105, 155 are program storage devices. The software is executed using the processors 110, 160 in a known manner.

[0014] Fig. 2 illustrates a method for providing version control. At block 205, the client-side API method is invoked at the client host 100, passing in a set of parameter objects (block 210). At block 215, a command object is provided that contains the parameter objects. At block 220, the command object and parameter objects are

serialized by the client host's code, such as Java code. At block 225, the serialized objects are sent from the client host 100 to the server host 150 over the network 130. The server host 150 receives the serialized parameters and, at block 230, attempts to deserialize them. The serialized command object is also received and deserialized. If the server host 150 can't successfully deserialized all of the parameter objects (block 235), e.g., the Class type of one or more of the parameters is not known by the server host 150, it is concluded that the server host 150 does not support the one or more parameters that cannot be successfully deserialized (block 240). In this case, the server host 150 may send an error message to the client host 100 informing it that the one or more parameters are not supported. The client host 100 may then decide, e.g., to reformulate its command so that the one or more unsupported parameters are not included, or to locate another server host that may be compatible with the one or more parameters in question. If the server host 150 can successfully deserialize all of the parameter objects (block 250), it carries out the command, including the functions represented by the parameters (block 255).

[0015] Specifically, in accordance with the invention, each new parameter for a command based API is defined as a unique new Java Class type. Each parameter Class may or may not contain data. Java Serialization is used to transmit parameters from a Java Client, e.g., client host 100, to a Java Server, e.g., server host 150. Whenever a new parameter (class) is added to the client host 100, and the parameter is sent to the server host 150, the server host 150 will either deserialize the parameter successfully if the server supports the functionality represented by the parameter, or fail to successfully deserialize it, indicating that the server does not support the parameter/functionality. A particular advantage of this technique for providing version control is that only one definition for a parameter is required. The version control information and the functionality are contained within one object, so no additional, separate version control data is needed. The Class type itself is the only unique data necessary to determine

whether the API at the server host 150 supports a given functionality. Thus, an important aspect of the invention is how the classes for commands and parameters are organized. Example declarations of classes are provided further below.

[0016] Serialization is the process of converting an object to a byte stream. Deserialization is the opposite process. When an object is serialized, information about its class and other objects that it refers to is also saved. To serialize an object using Java, an object is passed as an argument to the `writeObject()` method of an object of class `java.io.ObjectOutputStream` which, in turn, is built from an object of class `java.io.FileOutputStream`. Moreover, an object is serializable when its class implements the `Serializable` interface. This is an empty interface that doesn't contain any method declarations, but simply identifies classes whose objects are serializable. During deserialization, the `readObject()` method is invoked on an object of class `java.io.ObjectInputStream`, which is built upon a `java.io.FileInputStream` object. If the server host 150 cannot locate a class file needed to make sense of a parameter object during deserialization, the exception `java.lang.ClassNotFoundException` is thrown.

[0017] Example code for performing serialization is as follows:

```
Class Command1 implements Serializable{
    Parameter[] parms;
    Public Command1(Parameter[] parms);
};
Class Parameter implements Serializable {};
Class ParameterA {} extends Parameter {
    Public static ParameterA Option = new ParameterA();
};
Class ParameterB {} extends Parameter{
    Public static ParameterB Option = new ParameterB();
}
```

// Example code to create command and serialize:

```
Parameter[] parms = {ParameterA.Option, ParameterB.Option}
```

```
Command1 commandInstance = new Command1(parms);
```

```
outputStream.writeObject(commandInstance);
```

[0018] Example code for performing deserialization is as follows:

```
Try {
```

```
Command1 command = (Command1) inputStream.readObject();
```

```
} catch (ClassNotFoundException) {
```

```
// This means the command is not supported
```

```
handleNotSupported();
```

```
}
```

[0019] Generally, the command serialization and parameter serialization are recursive. The process can be outlined as follows:

- 1) A Command contains Parameters.
- 2) Code invokes Command.writeObject().
- 3) Command.writeObject method knows that in order to serialize a Command object, it must serialize everything contained within that object.
- 4) So, then Command.writeObject() invokes Parameter.writeObject().
- 5) Parameter.writeObject() knows that in order to serialize a Parameter object, it must serialize everything contained within the Parameter.
- 6) All parameter data is serialized at this point. Parameter serialization is done.
- 7) Control turns back to Command.writeObject(). All Parameters have been serialized. Any other data in Command is serialized and Command Serialization completed.
- 8) Command.writeObject completes.

[0020] Moreover, the process may be analogized to nested Russian dolls. For instance, say we had three nested Russian dolls – big, medium and small. The top call would be to BigDoll.writeObject(), which under the covers would invoke, MediumDoll.writeObject(), which under the covers would invoke LittleDoll.writeObject(). Deserialization is similar (outside->in).

[0021] While the above examples are provided using Java™, which is a high-level object-oriented programming language developed by Sun Microsystems, Inc., the invention is suitable for use with other object-oriented programming languages as well.

[0022] The invention has been described herein with reference to particular exemplary embodiments. Certain alterations and modifications may be apparent to those skilled in the art, without departing from the scope of the invention. The exemplary embodiments are meant to be illustrative, not limiting of the scope of the invention, which is defined by the appended claims.